1st Asian Pacific Summer School on Formal Methods
Course 12: Static Analysis of C programs with Frama-C

Virgile Prevosto

CEA List

August 30, 2009

outline

Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

Motivations

Main objective

Statically determine some semantic properties of a program

- safety: pointer are all valid, no arithmetic overflow, ...
- termination
- functional properties
- dead code
- ...

Embedded code

- Much simpler than desktop applications
- Some parts are critical, *i.e.* a bug have severe consequences (financial loss, or even dead people)
- Thus a good target for static analysis

Some examples

- During first gulf war (1991), a patriot missile failed to intercept an Iraqi missile due to rounding errors
- Failure of Ariane 5 maiden flight (arithmetic overflows)
- ...

Some tools

Polyspace Verifier Checks for (absence of) run-time error C/C++/Ada)
`http://www.mathworks.com/products/polyspace/`

ASTRÉE Absence of error *without false alarm* in SCADE-generated code
`http://www.di.ens.fr/~cousot/projets/ASTREE/`

Coverity Checks for various code defects (C/C++/Java)
`http://www.coverity.com`

Some tools (cont'd)

a3 Worst-case execution time and Stack depth
`http://www.absint.com/`

FLUCTUAT Accuracy of floating-point computations and origin of rounding errors
`http://www-list.cea.fr/labos/fr/LSL/fluctuat/`

Frama-C A toolbox for analysis of C programs
`http://frama-c.cea.fr/`

A brief history

- 90's: CAVEAT, an Hoare logic-based tool for C programs
- 2000's: CAVEAT used by Airbus during certification process of the A380
- 2002: Why and its C front-end Caduceus
- 2006: Joint project to write a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C

A brief history

- 90's: CAVEAT, an Hoare logic-based tool for C programs
- 2000's: CAVEAT used by Airbus during certification process of the A380
- 2002: Why and its C front-end Caduceus
- 2006: Joint project to write a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C

A brief history

- 90's: CAVEAT, an Hoare logic-based tool for C programs
- 2000's: CAVEAT used by Airbus during certification process of the A380
- 2002: Why and its C front-end Caduceus
- 2006: Joint project to write a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C

frama C
Software Analyzers

A brief history

- 90's: CAVEAT, an Hoare logic-based tool for C programs
- 2000's: CAVEAT used by Airbus during certification process of the A380
- 2002: Why and its C front-end Caduceus
- 2006: Joint project to write a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C

A brief history

- 90's: CAVEAT, an Hoare logic-based tool for C programs
- 2000's: CAVEAT used by Airbus during certification process of the A380
- 2002: Why and its C front-end Caduceus
- 2006: Joint project to write a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Software Analyzers

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Architecture

- A modular architecture
- Kernel:
  - CIL (U. Berkeley) library for the C front-end
  - ACSL front-end
  - Global management of analyzer's state
- Various plug-ins for the analysis
  - Value analysis (abstract interpretation)
  - Jessie (translation to Why)
  - Slicing
  - Impact analysis
  - ...

Presentation

Data-flow Analysis

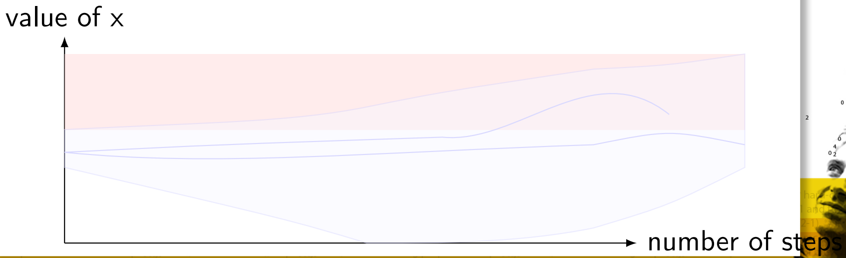
Abstract Interpretation

Abstract Interpretation in Practice

Concrete execution

Concrete semantics

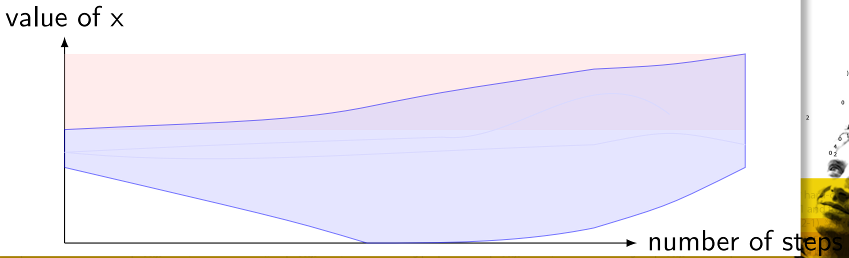
- Formalisation of all possible behaviors of a program
- Function which associate to a program an element of the concrete domain of interest
- Trace semantics: associate to each program point the values that the variables can take at this point



Concrete execution

Concrete semantics

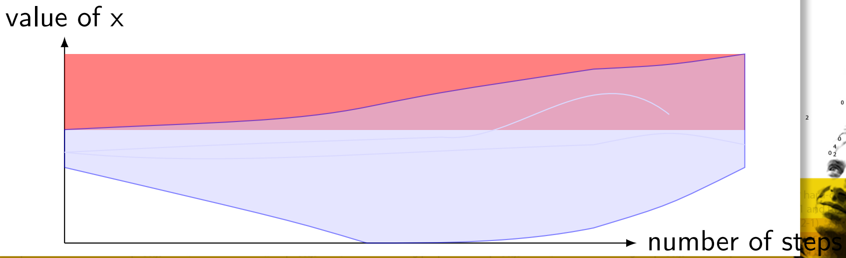
- Formalisation of all possible behaviors of a program
- Function which associate to a program an element of the concrete domain of interest
- Trace semantics: associate to each program point the values that the variables can take at this point



Concrete execution

Concrete semantics

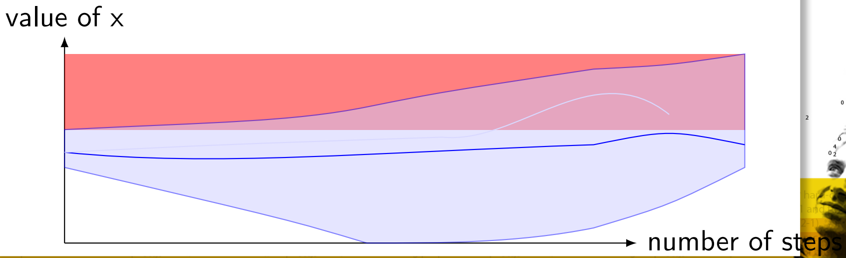
- Formalisation of all possible behaviors of a program
- Function which associate to a program an element of the concrete domain of interest
- Trace semantics: associate to each program point the values that the variables can take at this point



Concrete execution

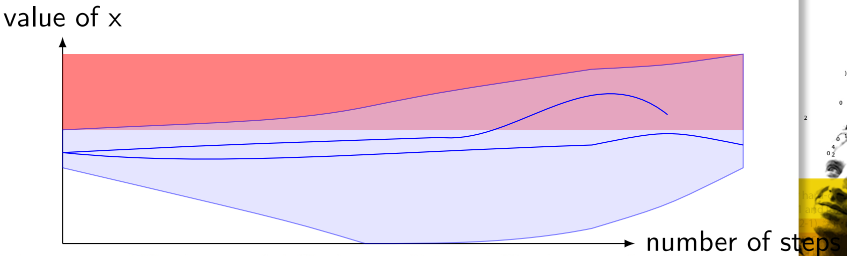
Concrete semantics

- Formalisation of all possible behaviors of a program
- Function which associate to a program an element of the concrete domain of interest
- Trace semantics: associate to each program point the values that the variables can take at this point



Concrete semantics

- Formalisation of all possible behaviors of a program
- Function which associate to a program an element of the concrete domain of interest
- Trace semantics: associate to each program point the values that the variables can take at this point



Example

```
int fact(int x) {
  int z = 1,y = 1;
  if (x<4) { x = 4; }
  while (y<=x) {
    z=z*y;
    y++;
  }
  return z;
}
```

Example

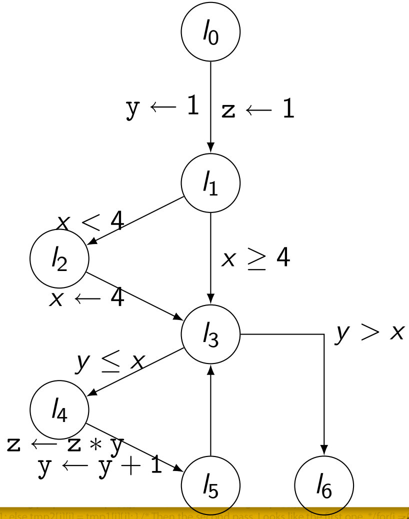
```
int fact(int x) {
  int z = 1,y = 1;
  if (x<4) { x = 4; }
  while (y<=x) {
    z=z*y;
    y++;
  }
  return z;
}
```

$z \in \{24, 120, 720, \ldots\}$

- Control-flow graph (CFG) of the program
- Each edge has an associated a transfer function $f_{i,j} : L \longrightarrow L$
- System of equations $l_i = \bigcup_{\{e_j \in l_j\}} \{f_{j,i}(e_j)\}$
- Solved by successive iterations (Kleene)

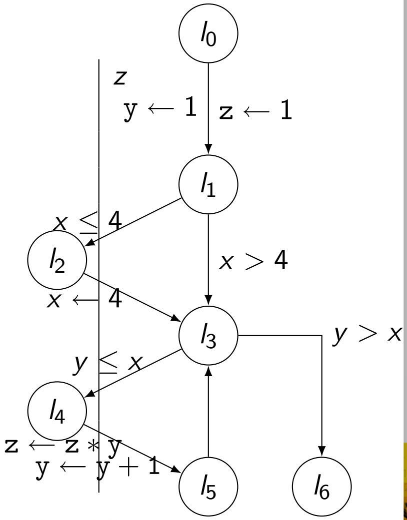
Control-flow Graph

```
int fact(int x) {   l0
  int z = 1,y = 1;   l1
  if (x<4) { x = 4;   l2 }
l3
  while (y<=x) {   l4
    z=z*y;
    y++;   l5
l6 }
  return z;
}
```



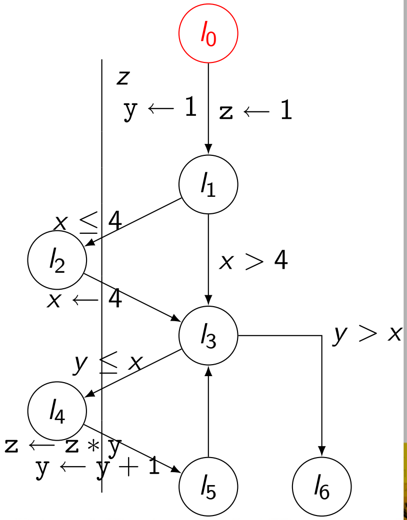
Iterations

	x	y	z
S_0			
S_1			
S_2			
S_3			
S_4			
S_5			
S_6			



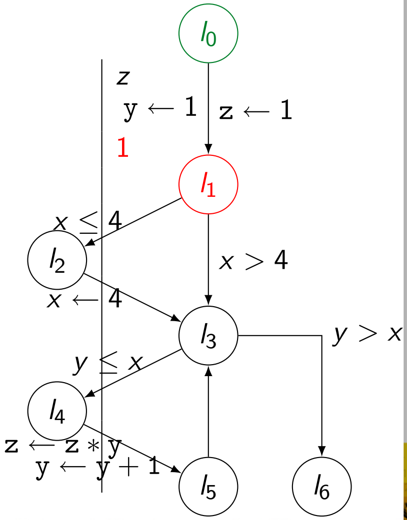
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1			
S_2			
S_3			
S_4			
S_5			
S_6			



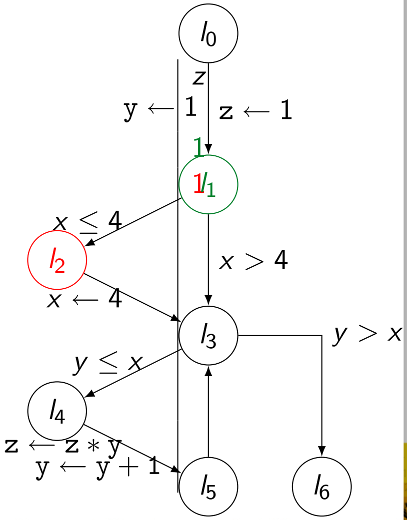
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2			
S_3			
S_4			
S_5			
S_6			



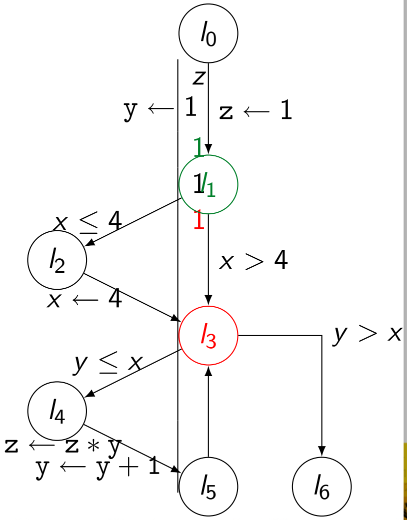
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3			
S_4			
S_5			
S_6			



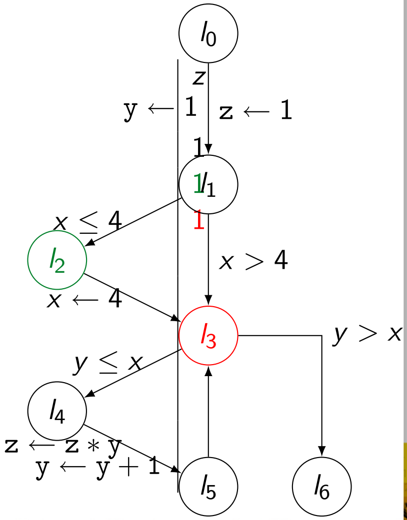
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4			
S_5			
S_6			



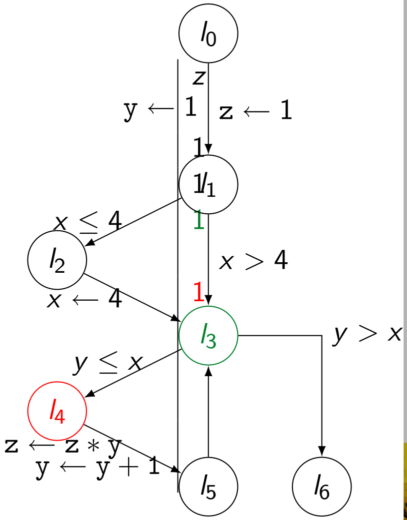
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4			
S_5			
S_6			

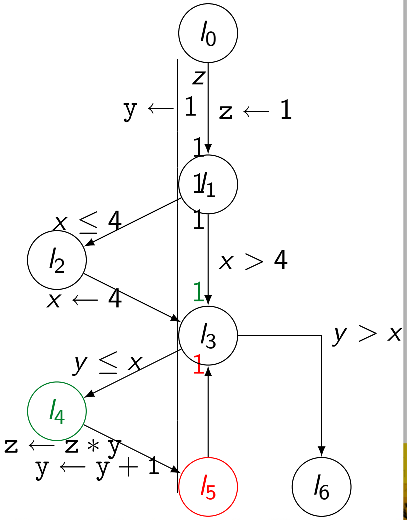


Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4	$4..+\infty$	1	1
S_5			
S_6			

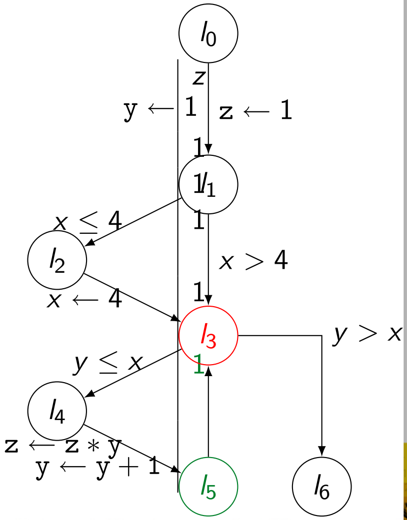


	x	y
S_0	$\mathbb{Z}$	
S_1	$\mathbb{Z}$	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1
S_4	$4..+\infty$	1
S_5	$4..+\infty$	2
S_6		



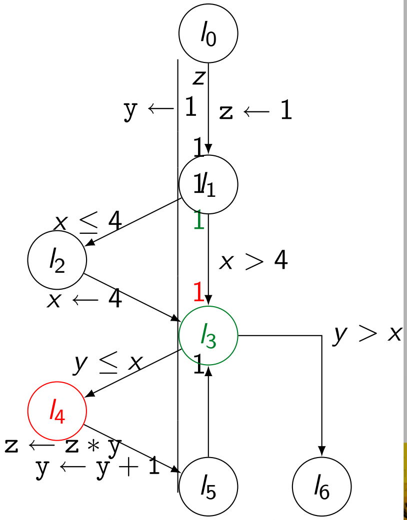
Iterations

	x	y
S_0	$\mathbb{Z}$	
S_1	$\mathbb{Z}$	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1,2
S_4	$4..+\infty$	1
S_5	$4..+\infty$	2
S_6		



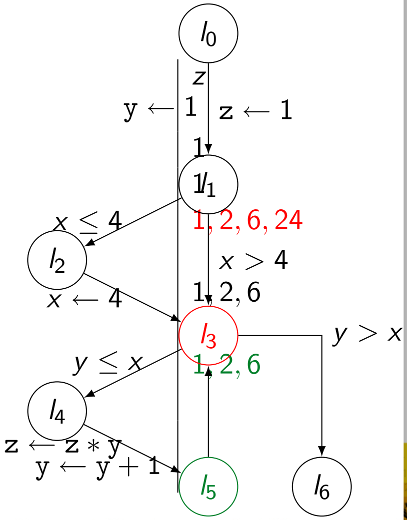
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1,2	1
S_4	$4..+\infty$	1,2	1
S_5	$4..+\infty$	2	1
S_6			



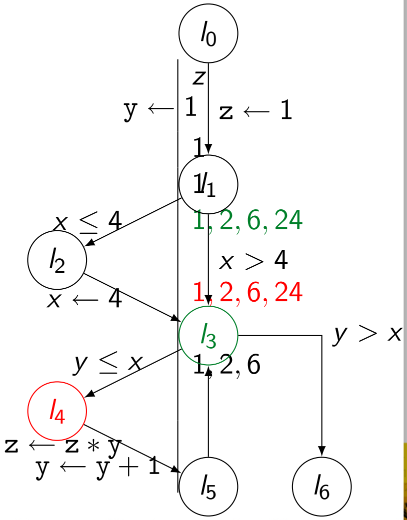
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..5$	1, 2, 6, 24
S_4	$4..+\infty$	$1..4$	1, 2, 6
S_5	$4..+\infty$	$2..5$	1, 2, 6
S_6			



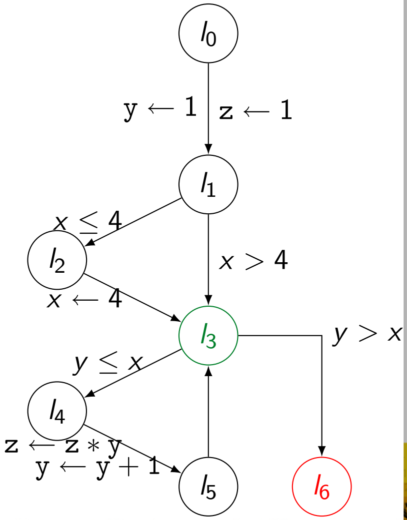
Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1..5	1, 2, 6, 24
S_4	$4..+\infty$	1..5	1, 2, 6, 24
S_5	$4..+\infty$	2..5	1, 2, 6
S_6			

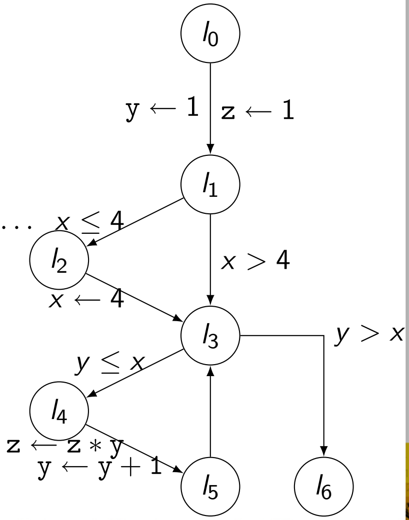


Iterations

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..5$	$1, 2, 6, 24$
S_4	$4..+\infty$	$1..5$	$1, 2, 6, 24$
S_5	$4..+\infty$	$2..5$	$1, 2, 6$
S_6	$4..+\infty$	5	24



	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1\ldots$	$1, 2, 6, 24\ldots$
S_4	$4..+\infty$	$1\ldots$	$1, 2, 6, \ldots$
S_5	$4..+\infty$	$2\ldots$	$1, 2, 6, \ldots$
S_6	$4..+\infty$	$4\ldots$	$1, 2, 6, \ldots$



Undecidability

Rice's Theorem

Any non-trivial semantic property of a program is undecidable.

Example

Halting problem: it cannot be decided statically if a given program will always terminate or not

Approximations

Even if the general case is unreachable, it is possible to devise analyses that give useful information

Undecidability

Rice's Theorem

Any non-trivial semantic property of a program is undecidable.

Example

Halting problem: it cannot be decided statically if a given program will always terminate or not

Approximations

Even if the general case is unreachable, it is possible to devise analyses that give useful information

Rice's Theorem

Any non-trivial semantic property of a program is undecidable.

Example

Halting problem: it cannot be decided statically if a given program will always terminate or not

Approximations

Even if the general case is unreachable, it is possible to devise analyses that give useful information

Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

Approximations

- Ensuring termination of the analysis
- Use abstract values
- Allows approximations
- may lead to false alarm

Abstract interpretation

- Formalized by Patrick and Radhia Cousot [POPL'77]
- Give relations between concrete and abstract domains (Galois connection)
- Termination (widening)
- Mixing information from distinct abstractions (reduced product)

Approximations

- Ensuring termination of the analysis
- Use abstract values
- Allows approximations
- may lead to false alarm

Abstract interpretation

- Formalized by Patrick and Radhia Cousot [POPL'77]
- Give relations between concrete and abstract domains (Galois connection)
- Termination (widening)
- Mixing information from distinct abstractions (reduced product)

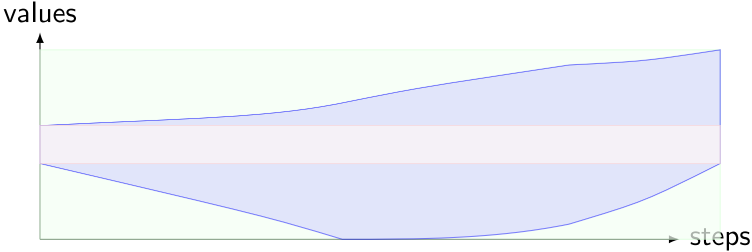
Correction and completeness

The approximation can be either

correct: All concrete behavior are represented by the abstraction

complete: All abstract behaviors are the representation of a concrete trace

but not both

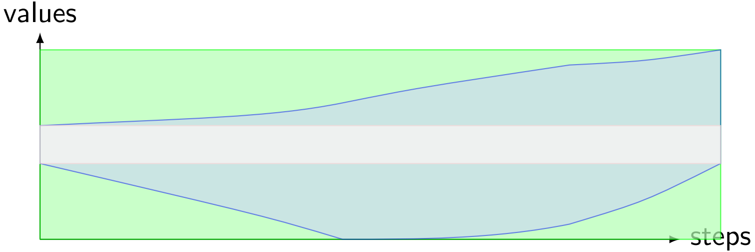


Correction and completeness

The approximation can be either

- **correct:** All concrete behavior are represented by the abstraction
- **complete:** All abstract behaviors are the representation of a concrete trace

but not both

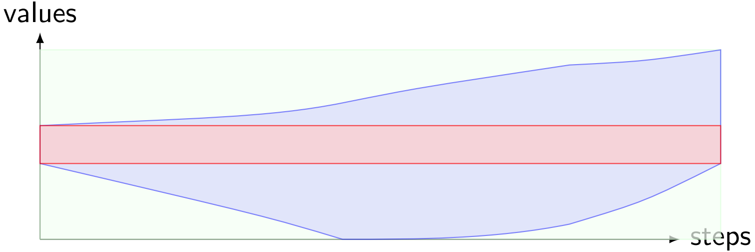


Correction and completeness

The approximation can be either

- correct: All concrete behavior are represented by the abstraction
- complete: All abstract behaviors are the representation of a concrete trace

but not both



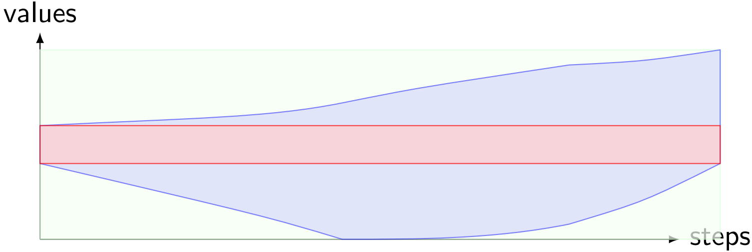
Correction and completeness

The approximation can be either

correct: All concrete behavior are represented by the abstraction

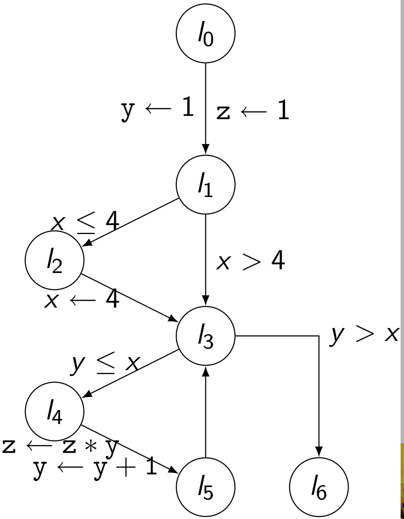
complete: All abstract behaviors are the representation of a concrete trace

but not both



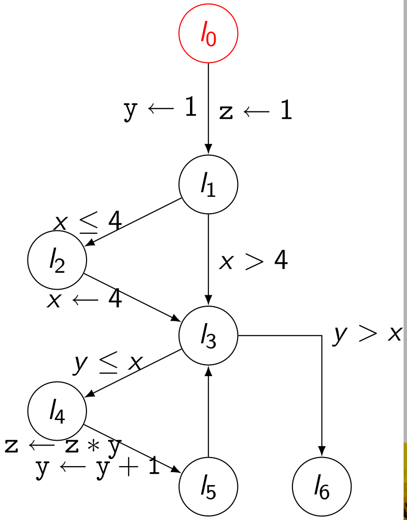
Example: intervals

	x	y	z
S_0			
S_1			
S_2			
S_3			
S_4			
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



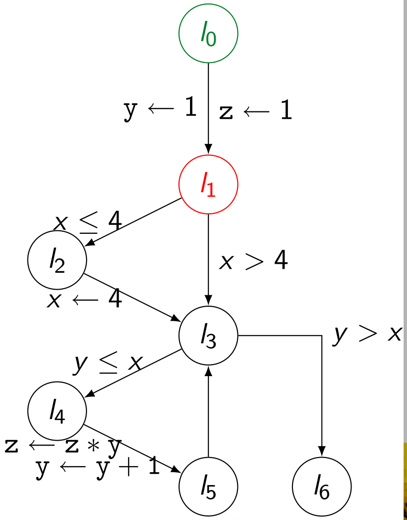
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1			
S_2			
S_3			
S_4			
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



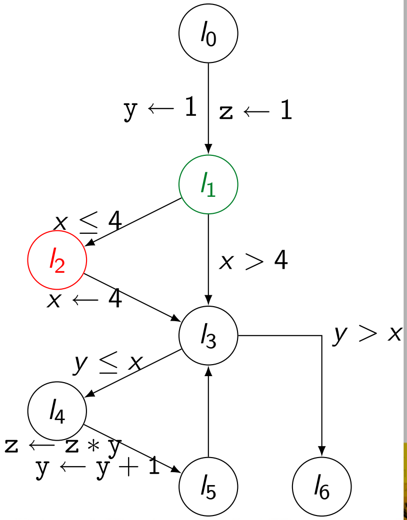
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2			
S_3			
S_4			
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



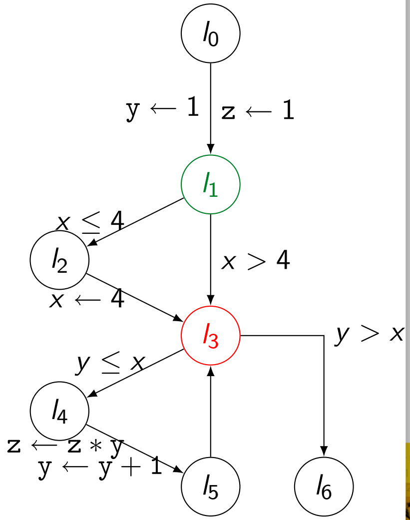
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3			
S_4			
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



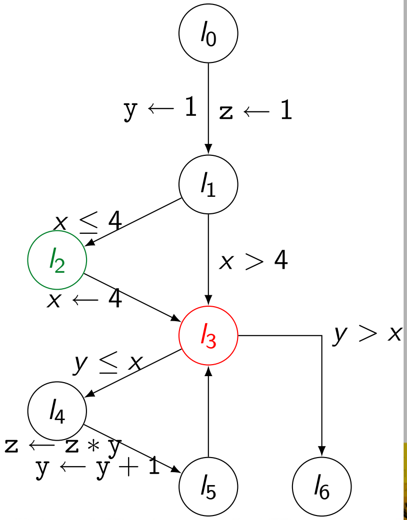
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4			
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



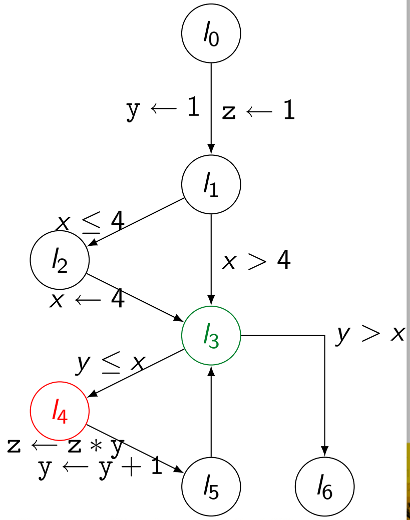
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4			
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



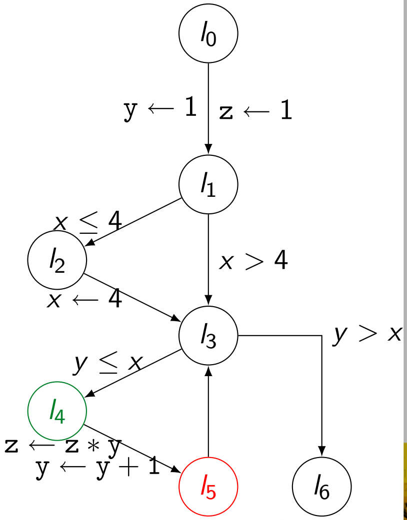
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4	$4..+\infty$	1	1
S_5			
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



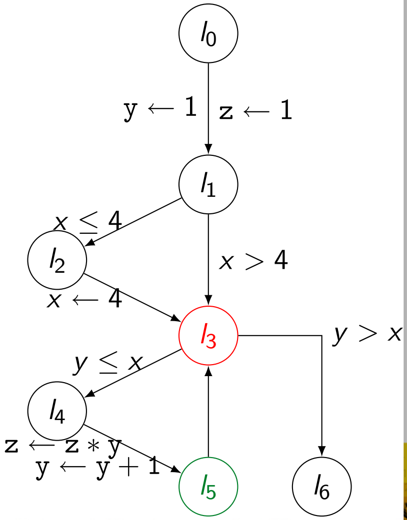
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4	$4..+\infty$	1	1
S_5	$4..+\infty$	2	1
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



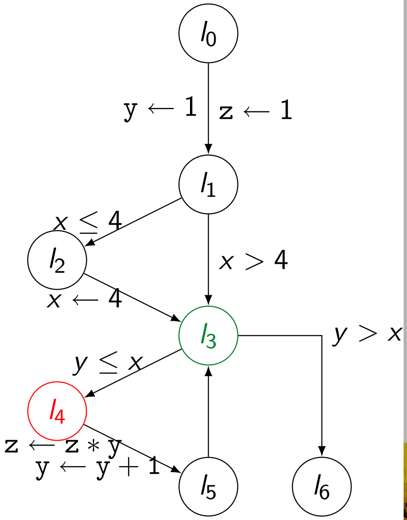
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	1
S_4	$4..+\infty$	1	1
S_5	$4..+\infty$	2	1
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



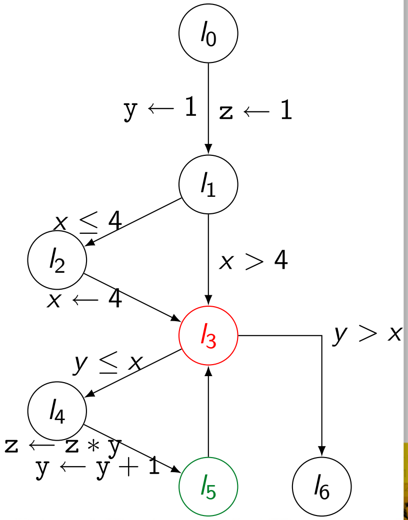
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	1
S_4	$4..+\infty$	$1..+\infty$	1
S_5	$4..+\infty$	2	1
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



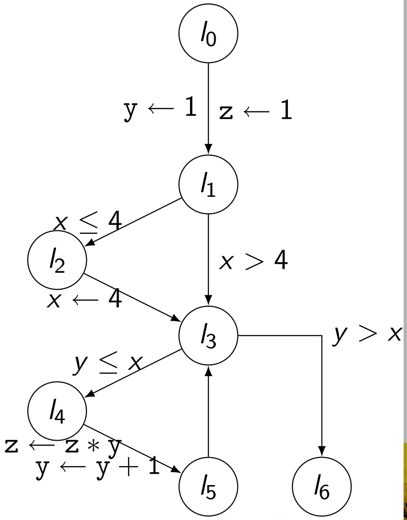
Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	$1..+\infty$
S_4	$4..+\infty$	$1..+\infty$	
S_5	$4..+\infty$	$2..+\infty$	$1..+\infty$
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



Example: intervals

	x	y	z
S_0	$\mathbb{Z}$		
S_1	$\mathbb{Z}$	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	$1..+\infty$
S_4	$4..+\infty$	$1..+\infty$	$1..+\infty$
S_5	$4..+\infty$	$2..+\infty$	$1..+\infty$
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$



Reduced product

Frama-c's integers abstraction uses intervals and modulo information.

- normal product: $x \in [0;8] \wedge x \equiv 1[2]$
- but $0 \not\equiv 1[2]$, so $x \in [1;7] \wedge x \equiv 1[2]$
- more generally the reduced product of two abstract domains allows to deduce more information than by doing two analyses separately

Frama-c's integers abstraction uses intervals and modulo information.

- normal product: $x \in [0;8] \wedge x \equiv 1[2]$
- but $0 \not\equiv 1[2]$, so $x \in [1;7] \wedge x \equiv 1[2]$
- more generally the reduced product of two abstract domains allows to deduce more information than by doing two analyses separately

Frama-c's integers abstraction uses intervals and modulo information.

- normal product: $x \in [0;8] \wedge x \equiv 1[2]$
- but $0 \not\equiv 1[2]$, so $x \in [1;7] \wedge x \equiv 1[2]$
- more generally the reduced product of two abstract domains allows to deduce more information than by doing two analyses separately

Frama-C Software Analyzers

Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

Abstract domains

Frama-C use mainly 3 domains:

- floating-point values: intervals
- integral types: intervals and modulo information
- pointers: set of possible base addresses + an offset (which is an integer).
- a few other refinements for pointed values

Representation of the state

In order to scale to realistic programs (100 kLOC or more), an efficient representation of the state of the program at each point is very important:

- Maximal sharing of the sub-expressions (hash-consing).
- Data structures allowing for fast search and insertion: variations over Patricia trees.
- some improvements of the Ocaml compiler itself have helped a lot.

Software Analyzers

Semantic unrolling

It is possible to regain some precision (at the expense of the performances) by keeping at most *n* states separated before merging or widening.

Example

```c
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

running frama-c

Semantic unrolling

It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$c \in [-\infty;\infty]$$
$$x \in [0;1]$$

running frama-c

Semantic unrolling

It is possible to regain some precision (at the expense of the performances) by keeping at most *n* states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$\begin{array}{lcr} c & \in &]-\infty;\infty[\\ x & \in & [0;1] \\ y & \in & [0;1] \end{array}$$

running frama-c

Semantic unrolling

It is possible to regain some precision (at the expense of the performances) by keeping at most *n* states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$\begin{array}{lcr} c & \in &]-\infty;\infty[\\ x & \in & [0;42] \\ y & \in & [0;42] \end{array}$$

running frama-c

Semantic unrolling

It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$
\begin{array}{lcl}
c & \in &]-\infty;0[\cup [0;\infty] \\
x & = & 0 \cup \quad 1
\end{array}
$$

running frama-c

Semantic unrolling

It is possible to regain some precision (at the expense of the performances) by keeping at most *n* states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$\begin{array}{rcl} c & \in &]-\infty;0[\cup[0;\infty[\\ x & = & 0\cup\quad 1 \\ y & = & 0\cup\quad 1 \end{array}$$

running frama-c

It is possible to regain some precision (at the expense of the performances) by keeping at most *n* states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$\begin{array}{lcrr} c & \in &]-\infty;0[\vee & [0;\infty[\\ x & = & 42\vee & 42 \\ y & = & 42\vee & 42 \end{array}$$

running frama-c

Exploiting value analysis results

It is possible to use the results of value analysis to produce more specialized results. This includes currently:

- semantic constant folding
- inputs and outputs of a function
- slicing
- impact analysis